Introduction

This is a technical audit for Sentivate token smart contract deployed on Ethereum blockchain platform. This documents outlines our methodology, limitations and results for our security audit.

Token name - Sentivate

Token Symbol - SNTVT

Decimals allowed - 18

SNTVT Token Total Supply - 4,200,000,000

Synopsis

Overall, the code demonstrates high code quality standards adopted and effective use of concept and modularity. Sentivate smart contract development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.

Code Analysis

Besides, the results of the automated analysis, manual verification was also taken into account. The complete contract was manually analysed, every logic was checked and compared with the one described in the whitepaper. The manual analysis of code confirms that the Contract does not contain any serious susceptibility. No divergence was found between the logic in Smart Contract and the whitepaper.

Scope

This audit is into the technical and security aspects of the Sentivate smart contract. The key aim of this audit is to ensure that tokens to be distributed to the investors are secure and calculations of the amount is exact. The next aim of this audit is to ensure the implementation

of token mechanism i.e. the Contract must follow all the ERC20 Standards. The audit of Smart Contract also checks the coded algorithms works as expected.

DevGenesis is one of the parties that independently audited Sentivate Smart Contract. This audit is purely technical and is not an investment advice. The scope of the audit is limited to the following source code file:

- **Filename:** Sentivate.sol
- **Github Repository:** https://github.com/shikhars371/Sentivate
- **Commit Hash:** f7a245f2311c4018bee7ba1c678d52d706fc466d

Limitations

Security auditing cannot bare all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a secure smart contract. However, auditing is there to find out vulnerabilities that were unobserved during development and areas where additional security measures are necessary. Some of the issues may affect the intact smart contract application, while some might lack protection only in certain areas. We therefore carry out a source code review to determine all locations that need to be fixed. DevGenesis has performed widespread auditing in order to discern as many susceptibilities as possible.

Traditional Way of Software Development

The code was provided to the auditors on Github. The codebase was properly version controlled. The code is written for Solidity version 0.4.18 and above.

The codebase uses community administered high quality Open Zeppelin framework. This software development practices and components match the expected community standards.

Sentivate Smart Contract Address: 0x7865af71cf0b288b4E7F654f4F7851EB46a2B7F8

Solidity Code:

```
/**
 *Submitted for verification at Etherscan.io on 2018-07-14
*/

pragma solidity ^0.4.18;

// ----------------------------------------------------------------------------
// 'SNTVT' token contract
//
// Deployed to : 0x2f3a5962357058E89FD6C86890d3d0b22b8983B1
// Symbol      : SNTVT
// Name        : Sentivate
// Total supply: 420000000000000000000000000
// Decimals    : 18
// ----------------------------------------------------------------------------


// ----------------------------------------------------------------------------
// Safe maths
// ----------------------------------------------------------------------------
contract SafeMath {
    function safeAdd(uint a, uint b) public pure returns (uint c) {
        c = a + b;
        require(c >= a);
    }
    function safeSub(uint a, uint b) public pure returns (uint c) {
        require(b <= a);
        c = a - b;
    }
    function safeMul(uint a, uint b) public pure returns (uint c) {
        c = a * b;
        require(a == 0 || c / a == b);
    }
    function safeDiv(uint a, uint b) public pure returns (uint c) {
        require(b > 0);
        c = a / b;
    }
}
```

```
// ----------------------------------------------------------------------------
// ERC Token Standard #20 Interface
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
// ----------------------------------------------------------------------------
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns
 (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (
bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint
 tokens);
}


// ----------------------------------------------------------------------------
// Contract function to receive approval and execute function in one call
//
// Borrowed from MiniMeToken
// ----------------------------------------------------------------------------
contract ApproveAndCallFallBack {
    function receiveApproval(address from, uint256 tokens, address token, bytes
 data) public;
}
```

```
// ----------------------------------------------------------------------------
// Owned contract
// ----------------------------------------------------------------------------
contract Owned {
    address public owner;
    address public newOwner;

    event OwnershipTransferred(address indexed _from, address indexed _to);

    function Owned() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function transferOwnership(address _newOwner) public onlyOwner {
        newOwner = _newOwner;
    }
    function acceptOwnership() public {
        require(msg.sender == newOwner);
        OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = address(0);
    }
}
```

```
// ----------------------------------------------------------------------------
// ERC20 Token, with the addition of symbol, name and decimals and assisted
// token transfers
// ----------------------------------------------------------------------------
contract Sentivate is ERC20Interface, Owned, SafeMath {
    string public symbol;
    string public  name;
    uint8 public decimals;
    uint public _totalSupply;

    mapping(address => uint) balances;
    mapping(address => mapping(address => uint)) allowed;


    // ------------------------------------------------------------------------
    // Constructor
    // ------------------------------------------------------------------------
    function Sentivate() public {
        symbol = "SNTVT";
        name = "Sentivate";
        decimals = 18;
        _totalSupply = 420000000000000000000000000;
        balances[0x2f3a5962357058E89FD6C86890d3d0b22b8983B1] = _totalSupply;
        Transfer(address(0), 0x2f3a5962357058E89FD6C86890d3d0b22b8983B1, _totalSupply);
    }


    // ------------------------------------------------------------------------
    // Total supply
    // ------------------------------------------------------------------------
    function totalSupply() public constant returns (uint) {
        return _totalSupply  - balances[address(0)];
    }


    // ------------------------------------------------------------------------
    // Get the token balance for account tokenOwner
    // ------------------------------------------------------------------------
    function balanceOf(address tokenOwner) public constant returns (uint balance) {
        return balances[tokenOwner];
    }
```

```
// ------------------------------------------------------------------------
// Transfer the balance from token owner's account to to account
// - Owner's account must have sufficient balance to transfer
// - 0 value transfers are allowed
// ------------------------------------------------------------------------
function transfer(address to, uint tokens) public returns (bool success) {
    balances[msg.sender] = safeSub(balances[msg.sender], tokens);
    balances[to] = safeAdd(balances[to], tokens);
    Transfer(msg.sender, to, tokens);
    return true;
}


// ------------------------------------------------------------------------
// Token owner can approve for spender to transferFrom(...) tokens
// from the token owner's account
//
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
// recommends that there are no checks for the approval double-spend attack
// as this should be implemented in user interfaces
// ------------------------------------------------------------------------
function approve(address spender, uint tokens) public returns (bool success) {
    allowed[msg.sender][spender] = tokens;
    Approval(msg.sender, spender, tokens);
    return true;
}


// ------------------------------------------------------------------------
// Transfer tokens from the from account to the to account
//
// The calling account must already have sufficient tokens approve(...)-d
// for spending from the from account and
// - From account must have sufficient balance to transfer
// - Spender must have sufficient allowance to transfer
// - 0 value transfers are allowed
// ------------------------------------------------------------------------
function transferFrom(address from, address to, uint tokens) public returns (bool success) {
    balances[from] = safeSub(balances[from], tokens);
    allowed[from][msg.sender] = safeSub(allowed[from][msg.sender], tokens);
    balances[to] = safeAdd(balances[to], tokens);
    Transfer(from, to, tokens);
    return true;
}


// ------------------------------------------------------------------------
// Returns the amount of tokens approved by the owner that can be
// transferred to the spender's account
// ------------------------------------------------------------------------
function allowance(address tokenOwner, address spender) public constant returns (uint remaining) {
    return allowed[tokenOwner][spender];
}


// ------------------------------------------------------------------------
// Token owner can approve for spender to transferFrom(...) tokens
// from the token owner's account. The spender contract function
// receiveApproval(...) is then executed
// ------------------------------------------------------------------------
function approveAndCall(address spender, uint tokens, bytes data) public returns (bool success) {
    allowed[msg.sender][spender] = tokens;
    Approval(msg.sender, spender, tokens);
    ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this, data);
    return true;
}


// ------------------------------------------------------------------------
// Don't accept ETH
// ------------------------------------------------------------------------
function () public payable {
    revert();
```

```
    }


    // ------------------------------------------------------------------------
    // Owner can transfer out any accidentally sent ERC20 tokens
    // ------------------------------------------------------------------------
    function transferAnyERC20Token(address tokenAddress, uint tokens) public onlyOwner returns (bool success) {
        return ERC20Interface(tokenAddress).transfer(owner, tokens);
    }
}
```

Overview

The project has only one file, the Sentivate.sol file which contains 223 lines of Solidity code. All the functions and state variables are well commented using the Natspec documentation for the functions which is good to understand quickly how everything is supposed to work.

Testing

Sentivate smart contract went through rigorous testing, which focused on the security features of the smart contract. During the complete Test phase the security of the Project was prime consideration. Major Task was to find and describe the security issues in the Smart Contract.

Primary checks followed during testing of Smart Contract is to see that if code :

- We check the Smart Contracts Logic and compare it with one described in the Whitepaper.
- The contract code should follow the Conditions and logic as per user request.

- We deploy the Contract and run the Tests.
- We make sure that the Contract does not lose any money/Ether.

Vulnerabilities Check

Smart Contract was scanned for commonly known and more specific vulnerabilities.

Following are the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- **TimeStamp Dependence:** The timestamp of the block can be manipulated by the miner, and so should not be used for critical components of the contract. *Block numbers* and *average block time* can be used to estimate time (suggested). Sentivate smart contract does not have any timestamp dependence in its code.

- **Gas Limit and Loops:** Loops that do not have a fixed number of iterations, hence due to normal operation, the number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Sentivate smart contract is free from the gas limit check as the contract code does not contain any loop in its code.

- **Compiler Version:** Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Sentivate smart contract is locked to a specific compiler version of 0.4.18 which is good coding practice.

- **ERC20 Standards:** Sentivate smart contract follows all the universal ERC20 coding standards and implements all its functions and events in the contract code.

- **Redundant fallback function:** The standard execution cost of a fallback function should be less than 2300 gas, Sentivate smart contract code has a fallback function and cost of its execution is less.

- **Unchecked math:** Need to guard uint overflow or security flaws by implementing the proper math logic checks. The Sentivate smart contract uses the popular SafeMath library for

critical operations to avoid arithmetic over or underflows and safeguard against unwanted behaviour. In particular, the 'balances' variable is updated using the safemath operation.

- **Exception disorder:** When an exception is thrown, it cannot be caught: the execution stops, the fee is lost. The irregularity in how exceptions are handled may affect the security of contracts.

- **Unsafe type Inference:** It is not always necessary to explicitly specify the type of a variable, the compiler automatically infers it from the type of the first expression that is assigned to the variable.

- **Reentrancy:** The reentrancy attack consists of the recursively calling a method to extract ether from a contract if user is not updating the balance of the sender before sending the ether.
  In Sentivate smart contract calls to external functions happen after any changes to state variables in the contract so the contract is not vulnerable to a reentrancy exploit. The Sentivate smart contract does not have any vulnerabilities against reentrancy attack.

- **DoS with (Unexpected) Throw:** The Contract code can be vulnerable to the Call Depth Attack! So instead, code should have a pull payment system instead of push. The Sentivate smart contract does not implement any payment related scenario thus it is not vulnerable to this attack.

- **DoS with Block Gas Limit:** In a contract by paying out to everyone at once, contract risk running into the block gas limit. Each Sentivate block can process a certain maximum amount of computation. If one try to go over that, the transaction will fail. Therefore again push over pull payment is suggested to remove the above issue.

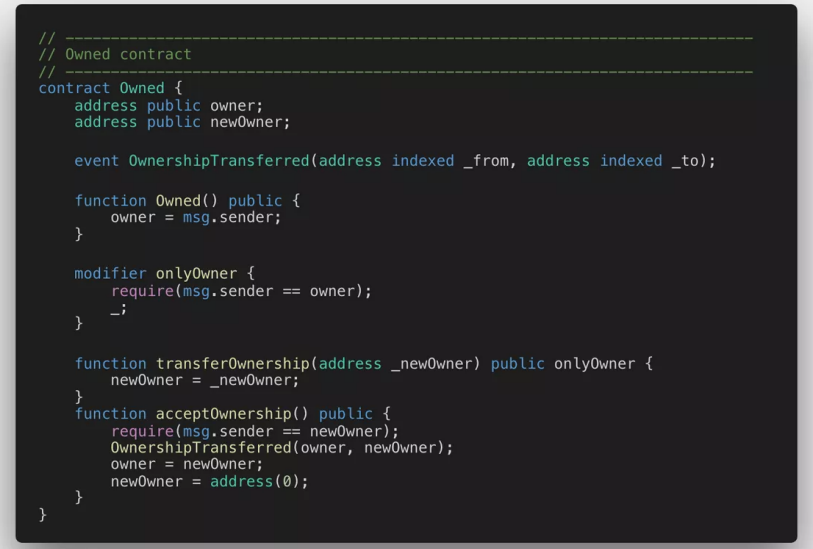
- **Explicit Visibility in functions and state variables:** Explicit visibility in the function and state variables are provided. Visibility like external, internal, private and public is used and defined properly.

Features in Smart Contract

1. **Ownable -** Sentivate Smart Contract inherit Owned contract. The smart contract i.e implemented i.e Sentivate -- are owned by a particular entity (ethereum address). To use certain function in Smart-contract, owner will have to call that function with his private key.

As we deploy the smart contract on Ethereum blockchain, smart-contract will be owned by the ethereum address who deploy the Sentivate Smart-Contract. Below is a code snippet showing Owned Smart-contract which is inherited by Sentivate Smart-Contract.

Code: Line 71 - 95



```
// ----------------------------------------------------------------------------
// Owned contract
// ----------------------------------------------------------------------------
contract Owned {
    address public owner;
    address public newOwner;

    event OwnershipTransferred(address indexed _from, address indexed _to);

    function Owned() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function transferOwnership(address _newOwner) public onlyOwner {
        newOwner = _newOwner;
    }
    function acceptOwnership() public {
        require(msg.sender == newOwner);
        OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = address(0);
    }
}
```

2. **Transferrable -** The tokens can be transferred from one entity to other(like to exchanges for trading). This transfer can be made by using any ethereum wallet which supports ERC20 token standard, for eg. MyEtherWallet, Mist Etc.

   Code: Line 146 - 151

```
// ------------------------------------------------------------------------
// Transfer the balance from token owner's account to to account
// - Owner's account must have sufficient balance to transfer
// - 0 value transfers are allowed
// ------------------------------------------------------------------------
function transfer(address to, uint tokens) public returns (bool success) {
    balances[msg.sender] = safeSub(balances[msg.sender], tokens);
    balances[to] = safeAdd(balances[to], tokens);
    Transfer(msg.sender, to, tokens);
    return true;
}
```

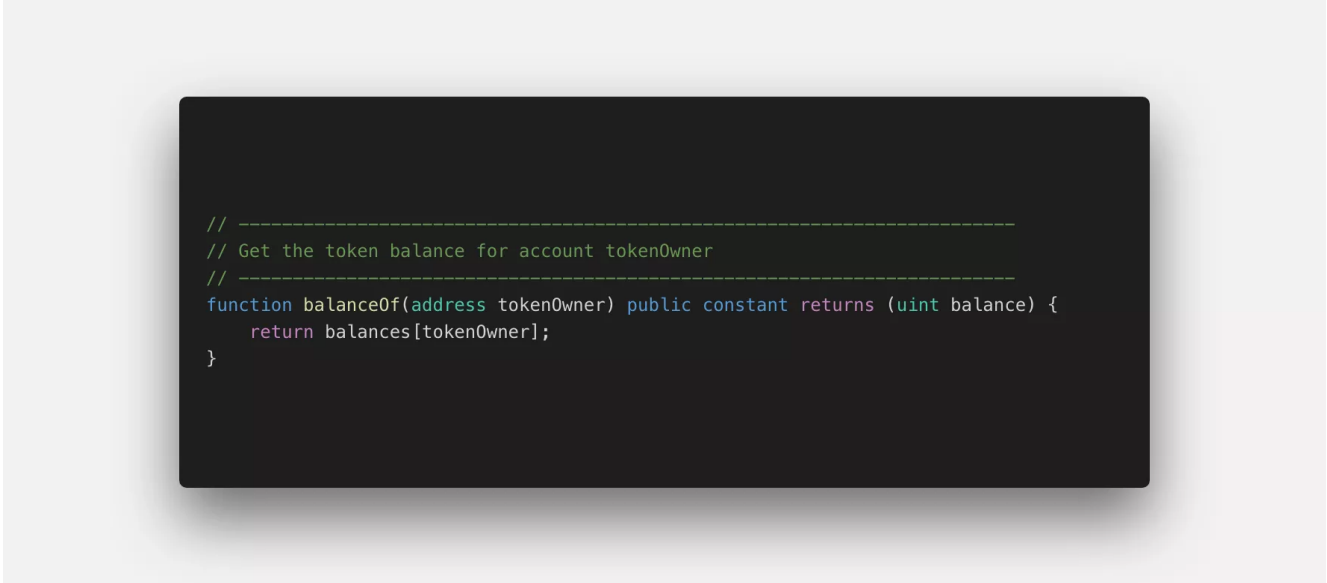
3. **Approvable -** Any Token holder if he wills, can approve some other address, who will on his behalf and the approved entity can transfer the approved amount of tokens from token holder's to others.

   Code: Line 162 - 166

```
// ------------------------------------------------------------------------
// Token owner can approve for spender to transferFrom(...) tokens
// from the token owner's account
//
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
// recommends that there are no checks for the approval double-spend attack
// as this should be implemented in user interfaces
// ------------------------------------------------------------------------
function approve(address spender, uint tokens) public returns (bool success) {
    allowed[msg.sender][spender] = tokens;
    Approval(msg.sender, spender, tokens);
    return true;
}
```

4. **Viewable Tokens -** As you already may be aware with the transparent nature of blockchain, all the tokens holders and their exact balance is made clearly visible, on Ethereum blockchain explorers like Etherscan and Ethplorer. Their are functions which are implemented to return informations like token balance of any particular token holder, the token allowance amount of any particular Token holder, which he has allowed to any other entity(Ethereum address).

Code: **Line 136 - 138**



```
// ------------------------------------------------------------------------
// Get the token balance for account tokenOwner
// ------------------------------------------------------------------------
function balanceOf(address tokenOwner) public constant returns (uint balance) {
    return balances[tokenOwner];
}
```

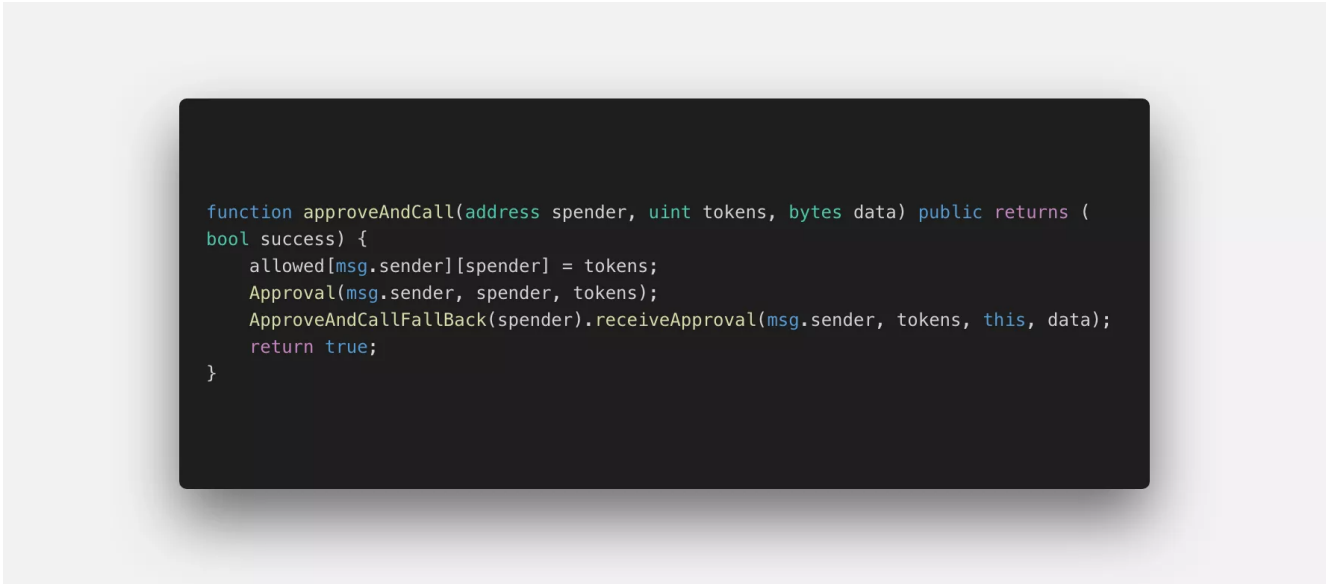
5. **Transfer Ownership & Accept Ownership -** Ownership of the smart contract can be transferred to a new Ethereum address(entity), this can be done only by the current owner of the smart contract, New owner should call a function of acceptOwnership() to accept the ownership of the Smart-Contract.

Code: **Line 86 - 94**

```
function transferOwnership(address _newOwner) public onlyOwner {
    newOwner = _newOwner;
}
function acceptOwnership() public {
    require(msg.sender == newOwner);
    OwnershipTransferred(owner, newOwner);
    owner = newOwner;
    newOwner = address(0);
}
```

6. **Approve and Call -** ERC20 requires a multistep process for tokens to be transferred to a contract. First approve must be called on the token contract, enabling the contract to withdraw the tokens. Next, the contract needs to be informed that it has been approved to withdraw tokens. Finally, the contract has to actually withdraw the tokens, and run any code related to receiving tokens. This process typically takes two to three steps, which is inefficient and a poor user experience. This issue is solved by approve and call function.

   Code: Line 201 - 206



```
function approveAndCall(address spender, uint tokens, bytes data) public returns (
bool success) {
    allowed[msg.sender][spender] = tokens;
    Approval(msg.sender, spender, tokens);
    ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this, data);
    return true;
}
```

7. **FallBack Function -** A Solidity contract may have a single unnamed function, no more no less. This functions cannot have any arguments, nor return anything.This Contract does-not accept any ether,hence revert() is implemented inside fallback function.

   Code: Line 212 - 214

```
// ----------------------------------------------------------------------------
// Don't accept ETH
// ----------------------------------------------------------------------------
function () public payable {
    revert();
}
```

8. **transferAnyERC20Token -** This Smart contract implemented a function named transferAnyERC20Token() which is called by only owner of Sentivate Smart-Contract in case if any token comes at the address of Sentivate Smart-Contract. As the name - owner transfers any token which is present on smart-contract, to any address which he wants.

   Code: Line 220 - 222

```
// ------------------------------------------------------------------------
// Owner can transfer out any accidentally sent ERC20 tokens
// ------------------------------------------------------------------------
function transferAnyERC20Token(address tokenAddress, uint tokens) public onlyOwner returns (bool success) {
    return ERC20Interface(tokenAddress).transfer(owner, tokens);
}
```

Risk

The Sentivate Smart Contract has no the risk of losing any amounts of ethers or tokens in case of external attack or a bug, as contract does not takes any kind of funds from the user. If anyone tries to send any amount of ether to the contract address, the transaction will cancel itself and no ether comes to the contracts.

The flow of tokens from this Sentivate contract can be controlled using a script running on the backend and visually through Etherscan.io. By using Etherscan.io working of code can be verified which will lead the compiled code getting matched with the bytecode of deployed smart contract in the blockchain.

Therefore, there is no anomalous gap in the Sentivate smart contract, tokens will be distributed to all the investors as per the amount paid by them during Pre-ICO/ ICO. As all the funds are held with owner's address thus he will be distributing all the tokens, so any possible losses due to flaws in the Sentivate smart contract is not possible to occur.

Conclusion

In this report, we have concluded about the security of Sentivate Smart Contract. The smart contract has been analysed under different facets. Code quality is very good, and well modularised. We found that Sentivate smart contract adapts a very good coding practice and have clean, documented code. Smart Contract logic was checked and compared with the one described in the whitepaper. No discrepancies were found.